Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	10	satellite and (tcp with spoofing)	USPAT	OR	ON	2005/04/29 15:15
L2	4	@ad<"20000721" and 1	USPAT	OR	ON	2005/04/29 15:16
L3	66	connection adj control adj block	USPAT	OR	ON	2005/04/29 15:16
L4	0	2 and 3	USPAT	OR	ON	2005/04/29 15:16
L5	35	tcp with spoofing	USPAT	OR	ON	2005/04/29 15:16
L6	0	3 and L5	USPAT	OR	ON	2005/04/29 15:17
S1	3748	709/217-219.ccls.	USPAT	OR	OFF	2005/04/29 10:15
S2	5220	709/227-235.ccls.	USPAT	OR	OFF	2005/04/29 10:15
S3	2345	370/229-235.ccls.	USPAT	OR	OFF	2005/04/29 10:19
S4	238	710/68.ccls.	USPAT	OR	OFF	2005/04/29 10:19
S5	10203	S1 S2 S3 S4	USPAT	OR	OFF	2005/04/29 10:21
S10	3075091	@ad<"20000721"	USPAT	OR	OFF	2005/04/29 10:22
S11	8800	S5 and S10	USPAT	OR	OFF	2005/04/29 10:22
S12	35	tcp with spoofing	USPAT	OR	ON	2005/04/29 10:22
S13	12	S11 and S12	USPAT	OR	ON	2005/04/29 10:22
S14	9	compression and S12	USPAT	OR	ON	2005/04/29 10:56
S15	9	channels with tcp with compression	USPAT	OR	ON	2005/04/29 15:15

Second Edition

Communication Networks
A First Course

Jean Walrand
University of California at Berkeley

WCB
McGraw-Hill

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Lisbon London Madrid
Mexico City Milan New Delhi Seoul Singapore Sydney Taipei Toronto

This book is for my wife Annie who made it possible and my children Isabelle and Julie who made it necessary.

WCB/McGraw-Hill

A Division of The McGraw-Hill Companies

COMMUNICATION NETWORKS

This book is printed on acid-free paper.

2 3 4 5 6 7 8 9 0 DOC/DOC 1 0 9

ISBN 0-256-17404-0

Vice president and editorial director:	*Kevin T. Kane*
Publisher:	*Tom Casson*
Executive editor:	*Elizabeth A. Jones*
Senior developmental editor:	*Kelley Butcher*
Marketing manager:	*John T. Wannemacher*
Project manager:	*Pat Frederickson*
Production supervisor:	*Heather D. Burbridge*
Designer:	*Michael Warrell*
Compositor:	*Techsetters, Inc.*
Typeface:	*10/12 Times Roman*
Printer:	*R. R. Donnelley & Sons Company*

Library of Congress Cataloging-in-Publication Data

Walrand, Jean.
Communication networks: a first course / Jean Walrand. — 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-256-17404-0
1. Computer networks. I. Title
TK5105.5.W35 1998
004.6—dc21 97-45604

Printed in the United States of America

http://www.mhhe.com

CHAPTER 8 Security and Compression

We have discussed how networks transport files or bit streams between host computers. In this chapter we explain two important services that complement the transport services: security and compression.

Computer systems that are part of a network are potential targets of a number of threats that we examine, together with basic protections, in Section 8.1. Many security mechanisms involve cryptographic methods that we describe in Section 8.2. We examine security systems in Section 8.3.

The transfer of audio, video, even data over a network is greatly facilitated by compression algorithms. Compression reduces the number of bits needed to encode the information. Section 8.4 discusses general concepts that are useful for understanding all compression algorithms. That section also explains algorithms used for data compression. We explain audio compression in Section 8.5 and video compression in Section 8.6. Complements contain details about cryptography and the theory of compression.

8.1 Threats and Protections

We classify the threats faced by systems into threats against computers (or servers), users, and documents. In each case we describe the threats and we discuss protection mechanisms.

8.1.1 *Threats against Computers*

In Table 8.1 we summarize the threats against computers and the corresponding protections. The threats against computer systems include physical attack, infection by harmful programs, and intrusion by illegitimate users. Harmful programs enter a computer either by a file copied by a computer's legitimate user or via the network without authorization. Harmful programs that copy themselves in other programs are called *viruses*. They are called *worms* if they copy themselves across the network.

To protect a system against such infection, we can use infection detection programs that look for known sequences of instructions in harmful programs or that monitor changes

TABLE 8.1 Main Threats against Computers and Their Protections

Threats	Protections
Physical attack	Physical security (e.g., locks)
Infection	Virus detection
Intrusion	Password control, firewalls

in the size of files. Another method for protecting the computer system is to prevent or restrict the writing to disk or to memory by some programs. Unfortunately, such restriction require a modification of the operating system in single-user systems.

An *intrusion* is an illegitimate access to a computer. Such intrusion typically occurs by guessing a legitimate user's password or by getting access to a password file. Other forms of intrusion include stealing a connection by impersonating a user on another machine, using a debugging trapdoor in a program, and exploiting a programming mistake—called a *hole*—that provides access to user privileges.

To protect against intrusion, users should choose passwords that are difficult to guess and, preferably, change them periodically. Moreover, we should transmit only encrypted passwords over the network. System administrators should be aware of trapdoors and holes and close them.

Firewalls are computers that screen the traffic that enters a computer network. A firewall is a system that monitors and restricts the packets that go through it. A gateway can be used between networks to limit their access. For instance, an organization may install a gateway between its network and the Internet, or between the personnel department's network and the employees' network. Two types of firewall systems can be set up. The first type is based on a packet filter. This filter screens packets typically based on their address. The difficulty with this approach is that it is easy to forge an address. The second type of firewall is an application level gateway. In such a system, the firewalls force packets to go through a gateway. The system can then restrict the applications that go across the gateway. For instance, only email might be authorized for communication with the rest of the world. One objective might be to prevent confidential files from leaking out. However, email can be used for file transfers, so this approach does not eliminate all undesirable transactions.

8.1.2 Threats against Users

Table 8.2 shows the threats against users and protections against them. The sender of a document may need to be *authenticated.* For instance, in electronic money transfers one needs to ascertain the identity of the person who sent the check. Similarly, the person who signs a document must be authenticated in a way that cannot be repudiated. A user who logs onto a computer system to access files may also need to be authenticated.

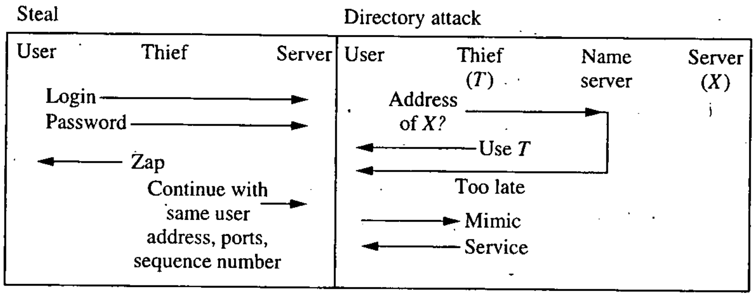
Authentication is required in some exchanges of information to prevent being fooled by an impersonator. Figure 8.1 shows two impersonation attacks. The first attack is the stealing of a connection by a "thief" who watches the login of a user. The thief then "zaps" the user by sending a barrage of messages to keep the user busy. The thief can then interact

TABLE 8.2 Main Threats against Users and Their Protections

Threats	Protections
Identity violation	Digital signature, encrypted password, watermarking
Privacy violation	Encryption, relays

FIGURE 8.1

Stealing a connection (left) and directory attack (right).



with the server by impersonating the legitimate user. The second attack is the "directory attack." Here, the thief replies to a request for an address by impersonating the name server. The thief can then mimic the server and get information from the user.

We explain in Section 8.3 methods that protect the integrity of data and of entities such as computers or users. Encryption is a family of methods for scrambling the messages so that eavesdroppers cannot read their contents. We discuss encryption in Section 8.2.

Watermarking is a method for including the identity of the author of a document. This method is the electronic equivalent of the watermark on paper currency. Watermarking is used to prevent another user from stealing a document and claiming authorship or ownership.

We say that *privacy* is violated when an eavesdropper discovers the identity of a user. To protect their identity, users may encrypt the messages they send. However, this encryption cannot hide the source of the messages. Users can also send their messages to a relay that resends them to their final destination. The relay may use confidential aliases for the destination addresses so that an eavesdropper cannot trace the (source, destination) pairs of messages.

Threats against Documents

Table 8.3 shows the threats against documents and protections against them. The *integrity* of a document is violated when the document is modified without authorization. Forging a signature, modifying the amount on a check, changing the name of an author are forms of integrity violation. Repudiating the authorship of a document is also possible when the integrity of documents is not guaranteed. The integrity of a document can be verified by

TABLE 8.3 Main Threats against Documents and Their Protections

Threats	Protections
Integrity violation	Message authentication code
Confidentiality violation	Encryption

including a message authentication code. *Confidentiality* of a document is violated when an unauthorized person discovers its content.

8.2 Cryptography

Cryptography, from the Greek, meaning "hidden writing," concerns the development of mechanisms that protect the contents of messages or the identity of their authors. We start by discussing the general principles of cryptography. We explain that two types of cryptography systems are used: secret key cryptography and public key cryptography. Cryptographic systems also use hashing functions that we introduce. Encryption and hashing are primitive operations that are used to build security systems that we examine in Section 8.3.

8.2.1 General Principles

In abstract terms, an encryption system works as indicated in Figure 8.2. The text that the sender—we call her Alice—must transmit in a secure way to the receiver Bob is the *plaintext* P. Alice converts P into the *ciphertext* C. Bob converts C back into P. Alice uses a function $E(.)$ to convert the plaintext. Bob recovers P from C by computing $D(C)$ where $D(.)$ is the inverse of the function $E(.)$. In this scheme, Alice and Bob must agree on the functions $E(.)$ and $D(.)$.

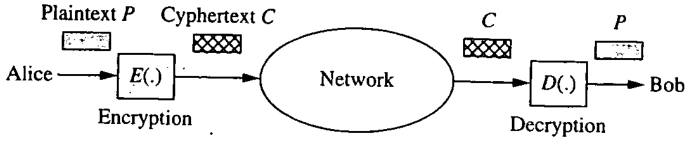
Eavesdroppers should find it difficult to identify the function D by looking at the messages that Alice sends. Experts have developed functions that have that desirable property. Before we examine them, let us discuss two examples of pair of functions E and D to illustrate how we could encode text.

A Naive Code

The *letter substitution code* replaces every letter with another one. For instance, one could have the correspondence $a \rightarrow u$, $b \rightarrow n$, $c \rightarrow f$, etc. With this code, the word $P = cab$ gets encoded into $C = fun$. The inverse operation converts $C = fun$ back into $P = cab$. This code is easily broken when used for plaintexts in English by using the relative frequencies of the letters. For instance, the letter e is the most frequently used in English. Thus, by finding the most frequent letter in the encoded text C, one knows that it should be decoded as e. One can continue in this way with the second most likely letter t, then the letter o, etc.

FIGURE 8.2

Encryption system. Alice encrypts her text P into C = E(P) before sending it over the network. Bob decrypts C into P = D(C).



One-Time Pad

We now describe a perfect secret code that is used in top secret military and diplomatic communication.

Alice wants to send a string P of n bits to Bob. Prior to the transmission, Alice and Bob agree on another n-bit sequence R whose bits are chosen independently of one another and are equally likely to be 1 or 0. Alice then sends $C = P \oplus R$ to Bob where the additions are bit-by-bit modulo 2. That is, Alice changes a bit of P whenever the corresponding bit of R is a 1. When Bob gets C, he computes $C \oplus R = P$.

This method is perfect in that the string C is perfectly random: its bits could have been produced by flipping a fair coin n times. Thus, C contains no information about P. Unfortunately, this method is not practical for most applications because R can be used only once for the scheme to be secure.

Communication systems use two types of cryptography: secret key cryptography and public key cryptography. We describe these two systems below.

8.2.2 *Secret Key Cryptography*

In secret key cryptography, Alice and Bob agree on a pair of secret functions E and D. That is, no one other than Alice and Bob knows which functions they use for their transmission.

The functions should be chosen so that even after observing many ciphertexts $C = E(P)$ it is difficult to reconstruct $E(.)$ and $D(.)$. For some applications, the functions should be difficult to reconstruct even after observing many pairs $\{P, E(P)\}$.

In practice, the functions $E(.)$ and $D(.)$ are a pair of functions $E(.; K)$ and $D(.; K)$. The algorithms to calculate E and D are known, but the *key* K is secret. The key is some n-bit binary word. For such a scheme to be secure, n should be large enough. Otherwise, a brute-force attack which consists in trying out systematically all the 2^n possible keys can succeed within a reasonable time and cost. For instance, $n = 40$ can be broken in a few hours by using a large number of computers on a network to try different keys. Current wisdom is that $n = 56$ bits is marginally secure, that 64 bits may not be secure against powerful organizations, and that $n = 128$ is probably secure for many decades. (Some experts will disagree, obviously.)

We should note that some secret key systems use secret algorithms $E(.; .)$ and $D(.; .)$. These algorithms are implemented in special-purpose integrated circuits that are difficult to reverse-engineer. In this situation, a smaller key may be secure, provided that reverse engineering does not become easier.

Of course, security is a probabilistic notion. An eavesdropper might be particularly lucky, but this event is exceedingly unlikely with a large n even if there are potentially many eavesdroppers interested in your files.

We explore specific secret key systems in Section 8.7.

8.2.3 Public Key Cryptography

In public key cryptography, Bob informs everyone that to send him messages one should use the encryption function $E(.)$, but he keeps the function $D(.)$ private. That is, to send the message P to Bob, Alice computes and transmits $E(P)$. Bob then calculates $D(E(P)) = P$.

In the public key systems we discuss, the functions E and D are inverse of each other. That is $E(D(P)) = D(E(P)) = P$. For such a scheme to be secure, $E(.)$ must be a *trapdoor one-way function*. By definition, a trapdoor one-way function $E(.)$ is one that admits an inverse $D(.)$ and that inverse must be very difficult to construct from the function $E(.)$ itself. Thus, there is little risk that anyone could construct $D(.)$ and start decrypting the messages that Alice sends to Bob.

Note that for a public key system to be usable, Alice needs to be confident that E is the public function of Bob, and not that of someone else pretending to be Bob. That is, one needs to trust the public keys. This certification is done in a hierarchical way.

As a rule, public key cryptography is numerically much more complex than secret key cryptography. That is, the calculations of $C = E(P)$ and of $P = D(C)$ take longer. Accordingly, many systems use secret key cryptography. We discuss the problem of distributing the secret keys in Section 8.7. We explore a specific public key system in Section 8.8.

8.2.4 Hashing

We explain below that cryptographic systems use *hash functions*. A hash function H maps a long text P into a shorter text $H(P)$. For instance, P might be 1000 bits long and $H(P)$ only 160 bits long. Thus, $H(.)$ is a many-to-one function: Many different texts P result in the same $H(P)$.

A hash function must be difficult (computationally) to pseudo-invert. That is, given M it is very difficult to construct a text P' such that $H(P') = M$. Depending of the application, the additional properties may be required:

- *Property 1:* Given both P and $H(P)$ it is difficult to construct $P' \neq P$ such that $H(P') = H(P)$. This condition is stronger than the previous one because knowing both P and $H(P)$ may make it easier to construct P' than if only $H(P)$ is known.
- *Property 2:* It is difficult to construct two texts P and $P' \neq P$ such that $H(P) = H(P')$.
- *Property 3:* Given P but not knowing a string K it is difficult to find $P' \neq P$ such that $H(K \circ P') = H(K \circ P)$. In this notation, $K \circ P$ means the concatenation of two strings K and P, that is, the bits of K followed by the bits of P. (For instance, $10011 \circ 011101 = 10011011101$.)

Our discussion of security systems indicates which properties are required of H.

8.3 Security Systems

In this section we explain security systems that combine encryption and hashing. We classify these systems by the services they provide. We conclude the section by looking at two widely used systems.

8.3.1 *Integrity*

We explain how cryptography can be used to protect the integrity of the data or of its origin. Imagine Alice sending a message P to Bob. She wants to guarantee that Bob gets P and not some other P'. Moreover, Bob wants to make sure that Alice sent the message, not someone else.

The basic idea of integrity protection systems is the following. To protect the integrity of a text P, Alice calculates $H(P)$ and sends $(P \circ H(P))$ to Bob. (Throughout, H is a hash function.) Note that $H(P)$ must be protected. Otherwise, an intruder Eve can intercept the transmission and send some $(P' \circ H(P'))$ to Alice. Three schemes are used to protect $H(P)$: authentic channel, message authentication code, and digital signature.

Authentic Channel

In the first scheme, Alice sends $H(P)$ to Bob over an *authentic channel.*. An authentic channel is defined as a channel whose transmissions cannot be corrupted. For instance, $H(P)$ could be communicated over a secure telephone line (whatever that means). The weakness of this scheme is that, given P, Eve might find some P' such that $H(P') = H(P)$. Eve could then intercept the unsecure transmission of P and replace it with P'. In this application, H should have property 2 described in Section 8.2 under "Hashing."

Message Authentication Code

In the second scheme, $H(P)$ is secured with secret keys. At first, one might think Alice could send P and $E(H(P); K)$ to Bob, using a secret key K. However, this scheme is not secure. For instance, imagine that Alice uses a one-time pad to encrypt $H(P)$. That is, Alice and Bob agree on a secret random string R and Alice sends $Z = R \oplus H(P)$. The intruder Eve sees P and Z and calculates $H(P)$ from P and then computes $H(P) \oplus Z = R$. Once she knows K, Eve can send P' and $H(P') \oplus R$ to Bob who then thinks Alice sent P'.

A suitable strategy is for Alice to send $H(K_2 \circ H(K_1 \circ P))$. That is, Alice first computes $V = H(K_1 \circ P)$, then $H(K_2 \circ V)$. Here, K_1 and K_2 are secret keys shared by Alice and Bob. This scheme fools Eve because she cannot calculate $H(K_1 \circ P')$. Accordingly, for this application, H should have property 2 of "Hashing" in Section 8.2. The string $H(K_2 \circ H(K_1 \circ P))$ is called a *message authentication code.*

Note that this discussion shows that the one-time pad is not secure for integrity applications although it is for confidentiality of the message.

Digital Signature

In this scheme, which uses public key encryption, Alice and Bob use the public function E of Alice whose inverse D is secret to Alice. The basic idea of the scheme is that Alice can sign a document P by sending $C = D(P)$ to Bob. Bob can recover P by calculating $P = E(C)$. (Recall that E and D are the inverse of one another.)

However, as described, the scheme is not suitable. Imagine that an intruder Eve intercepts the channel from Alice to Bob. Eve constructs a message C' and calculates $P' = E(C')$, using Alice's public key. Eve then sends C' to Bob who then calculates $P' = E(C')$ and might think that Alice sent $C' = D(P')$. Because of this possibility, this scheme should not be used: it does not protect the integrity of Alice.

A suitable scheme is for Alice to send $P \circ H(P)$ instead of P. That is, Alice sends $D(P \circ H(P))$ where H is a hash function. This makes it almost impossible for Eve to find another message C' so that $E(C')$ has the form $P' \circ H(P')$ for some text P'. The string $D(P \circ H(P))$ is said to be the *message P signed by* Alice.

Note that the digital signature is suitable even when Alice and Bob do not trust each other since they don't need to share secrets. Accordingly, a digital signature can be used for nonrepudiation of documents, which is critical in electronic commerce.

8.3.2 Key Management

Secret key systems require that Alice and Bob share a secret key K. Alice and Bob can use one of the following strategies to share such a secret key:

1. Hand delivery of the key.
2. Use a secret key to encrypt and distribute other secret keys. Kerberos uses this strategy (see "Kerberos," below).
3. Use a public key to establish secret keys. PGP uses this strategy (see "Pretty Good Privacy," below).
4. Public key agreement scheme. In such a scheme, Alice and Bob end up agreeing on a common key K by exchanging messages that intruders cannot use to compute K. The Diffie-Hellman exchange that we explain next is such a scheme.

Diffie-Hellman Exchange

This strategy is named after its inventors. Alice and Bob first agree on a pair of numbers (z, p). These numbers can be made public. Alice then chooses a number a and Bob chooses a number b. Alice computes $\alpha = z^a \bmod p$ and sends it to Bob who computes $\alpha^b \bmod p$. Similarly, Bob computes $\beta = z^b \bmod p$ and sends it to Alice who computes $\beta^a \bmod p$.

We claim that

$$\alpha^b \bmod p = \beta^a \bmod p = z^{ab} \bmod p =: K. \tag{8.1}$$

Moreover, it is difficult to calculate a from α and b from β, so that an eavesdropper who knows (z, p) and observes (α, β) cannot compute K.

To show (8.1), we first note that $\alpha = z^a \bmod p$ so that $\alpha = z^a + mp$ for some integer m. Consequently,

$$\alpha^b = (z^a + mp)^b = z^{ab} + bz^{a(b-1)}mp + \frac{b(b-1)}{2} z^{a(b-2)}(mp)^2 + \cdots + bz^a(mp)^{b-1} + (mp)^b.$$

This equality proves (8.1).

It turns out that the Diffie-Hellman exchange is not robust to a "person-in-the-middle" attack as we explain next. Imagine that an eavesdropper Eve intercepts the Diffie-Hellman exchange between Alice and Bob. Eve can communicate with Alice as if she were Bob and agree with Alice on a common secret key. In the same way, Eve can agree on a secret key with Bob as if she were Alice. At that point, Eve can intercept the messages from Alice to Bob, read them or change them and reencrypt them for Bob. Alice and Bob are unaware of the existence of Eve.

Another weakness of the Diffie-Hellman exchange is that Eve can intercept the message α from Alice to Bob and replace it by 0. Similarly, when the message β comes from Bob to Alice, Eve can replace it with 0. As a result, Alice and Bob end up agreeing on the common key 0 that Eve knows. (Eve could have used 1 or -1 instead of 0.) To prevent this simple attack, the system should verify that the numbers exchanged are not 0, 1, or -1.

A more subtle attack by Eve is as follows. Eve chooses some prime number q that divides $p-1$. She intercepts α from Alice to Bob and replaces it with α^q. She also intercepts β from Bob to Alice and replaces it with β^q. As a result, Alice and Bob end up agreeing on the key K^q instead of the key K. Although Eve does not know K^q, she knows that the key belongs to a smaller subgroup of keys (the exact qth powers) on which the search is considerably easier.

The Diffie-Hellman exchange can be made robust against this attack. Two solutions have been developed: signing the exchange and using safe primes.

When signing the exchange, Alice sends α to Bob and Bob sends β to Alice. Alice then signs (α, β) and sends it to Bob. Bob also signs (α, β) and sends it to Alice. Eve cannot fake the signatures.

By definition, a prime number p is a *safe prime* if $p' := (p-1)/2$ is also prime. In that case, one can show that the only multiplicative subgroups of $\{1, 2, \ldots, p-1\}$ are $\{1\}$, $\{-1, 1\}$, and $\{1, 2, \ldots, p-1\}$. It then suffices in the Diffie-Hellman exchange to check that the numbers exchanged are not 0, 1, or -1.

8.3.3 Identification

The identification problem is to ascertain the identity of some entity such as a computer or a user. In our discussion, Bob wants to ascertain the identity of Alice. We explain five mechanisms: passwords, challenge/response, public key, digital signature, and zero-knowledge proof.

Passwords

The simplest mechanism for identification is to use a password. To certify her identity, Alice has a secret password K and she sends (*Alice*, K) to Bob. Bob maintains a list of names with hashed passwords. This list contains (*Alice*, $H(K)$) and Bob can thus verify that Alice is who she claims to be. Bob does not keep the list of pairs (*Alice*, K) for obvious reasons. The weakness of the password mechanism is that Bob and the communication line both see the secret password K of Alice.

Challenge/Response

When using the *challenge/response* scheme, Bob sends a string X to Alice. Alice replies with a function $f(X, K)$ where K is a secret key that Bob and Alice share. Bob then checks the value of $f(X; K)$. If that value is correct, Bob assumes that Alice computed it since no one else knows K. The advantage of the scheme is that the key K is not transmitted over the line. The weakness is that Bob must know K.

Public Key
Let E be the public key of Alice with inverse D known only by Alice. Bob chooses X and sends $E(X)$ to Alice who decrypts $X = D(E(X))$ and sends X back to Bob.

Digital Signature
In this scheme, Bob sends X to Alice who signs it and returns the signed value to Bob.

Zero-Knowledge Proof
A zero-knowledge proof method enables a user to prove that it knows some specific information such as a secret password without revealing any of the information.

One zero-knowledge proof system is based on the notion of *quadratic residue*. Let x, y be positive integers that are relatively prime, i.e., that have no common factor other than 1. Say that y is a quadratic residue of x if there is some w such that $y = w^2 \bmod x$. For instance, 9 is a quadratic residue of 10 since $9 = 7^2 \bmod 10$. It turns out that, if x is hard to factor, then it is difficult to check whether a given y is a quadratic residue of x because it requires factoring x. Also, it is difficult to produce a "square root" w of a given y.

We show how Alice and Bob can use a zero-knowledge proof system to ascertain the identity of Alice.

- *Step 1:* Alice selects w (her password) and calculates $y = w^2 \bmod x$, for some large x. Alice gives y to Bob who then wants to make sure that Alice knows w. Bob cannot compute w from y.
- *Step 2:* Alice chooses some number u that is relatively prime with x and sends $z := u^2 \bmod x$ to Bob.
- *Step 3:* Bob sends back either 0 or 1 to Alice, with equal probabilities. If she gets 0, Alice must provide the value of u. If she gets 1, she must send $v := uw \bmod x$ to Bob. In the former case, Bob can check that $z = u^2 \bmod x$. In the second case, Bob can verify that $zy = v^2 \bmod x$.

An eavesdropper cannot recover w from either u or v alone, and the user cannot make up a value u when asked for a reply since the user does not know whether u or v will be asked. Also, without knowing w, Alice cannot produce a v such that $zy = v^2 \bmod x$ since it is difficult to compute square roots. Thus, Bob believes with probability $\frac{1}{2}$ that Alice knows w.

It is not trivial, but can be shown that repeated exchanges of this type do not reveal information about w.

8.3.4 *Replications and Deletions*

We have discussed cryptographic systems to protect the integrity and the confidentiality of data and their origin. Such schemes cannot protect a transmission channel against repetitions or deletions of messages. To protect against these corruptions, the system must use other strategies such as serial numbers for the messages.

and with a signature (A = Alice's public key):

$$[D(K; A), \text{MD5(Message)}].$$

PGP generates keys for you based on user ID, password, and random sequence of characters.

8.4 Foundations of Compression

In this section we explain why compression is possible and we explore some basic algorithms that compression methods use.

8.4.1 *Lossy and Lossless Compression*

Compression is possible because a source output contains *redundant* and/or *barely perceptible information.*

Redundant information is data that does not add information, such as white lines between lines of text on a page, periods of silence in a telephone signal, similar lines in a picture, and similar frames in a video sequence. Redundancy can be reduced by algorithms that then achieve a *lossless compression.* No information is lost by such algorithms even though they reduce the number of bits in the source output.

Barely perceptible information is, as the name indicates, information that does not affect the way we perceive the source output. Examples of barely perceptible information in audio are sounds at frequencies that the human ear does not hear and sounds that are masked by louder sounds. In images, some fine details are difficult to perceive and can be eliminated without much picture degradation. Such information is reduced or eliminated by *lossy compression algorithms.*

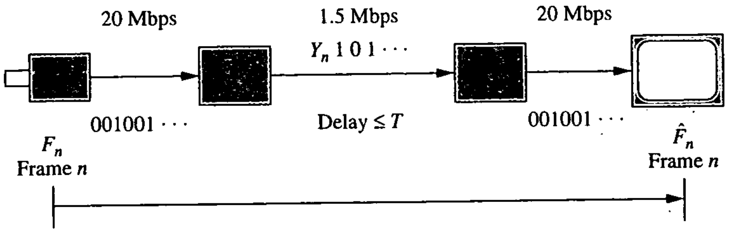
8.4.2 *Batch, Stream, Progressive, Multilayer*

Consider a source that produces a file F. We can view a compression algorithm as implementing a function $K(F)$ that produces a new file with fewer bits. The decompression algorithm calculates $X(K(F))$. (We use $K(.)$ for compress and $X(.)$ for expand, to avoid confusion with the cryptography notation.) This reconstruction must be exact in a lossless compression and, hopefully, it is almost exact in a lossy compression. That is, the distance $d(F, X(K(F)))$ between F and $X(K(F))$ is small in a lossy compression. We think of $d(.,.)$ as measuring a subjective perceptual difference, which is of course not easy to quantify.

The above formulation of the compression/decompression process ignores the timing of the operations. Consider the hypothetical compression/decompression of a video signal during a video conference shown in Figure 8.4. The video camera produces a bit stream at 20 Mbps. The compression algorithm converts this stream into a 1.5 Mbps that the decompression converts back into a 20-Mbps stream close to the original. More specifically, the video camera captures successive frames F_n at the rate of 30 frames per second. These frames are converted into a bit stream which is compressed, then decompressed. The receiver uses the decompressed bit stream to reconstruct frames $\hat{F}_n$ that approximate the

FIGURE 8.4

Delay in the compression/decompression of a video stream.



original ones. The delay between the capture of frame F_n and its reconstruction as $\hat{F}_n$ by the receiver is obviously of critical importance in real-time applications.

This delay is important also for stored video. Imagine that the compressed bit stream corresponding to 5 minutes of video is stored in a server. When a user wants to view that video clip, it is desirable for the playback to start as soon as possible. As we discussed in Chapter 4, the delay jitter in the network should be compensated for by a playback buffer. It is also clear that such a stream playback is possible only if the network can deliver the compressed bit stream without building up a backlog.

Similar considerations apply to audio.

In the case of (fixed) pictures, it is desirable to have a progressive reconstruction of the picture. That is, in trying to view an image that is stored as a large file, it is desirable to view quickly a low-resolution version of the image that improves progressively.

Finally, since network connections have widely different bandwidth, it is desirable to be able to request different versions of audio or video clips or streams. These versions correspond to different qualities of the reconstruction, being compressed differently. Multilayer compression algorithms provide complementary versions of a clip, like adding layers of details to a coarse initial version to provide increasingly fine reproductions. Researchers are experimenting with various network protocols that select adaptively the number of layers based on the delays.

8.4.3 Source Coding

We view the source as producing a string of symbols. These symbols may be pixels, sound samples in audio, or characters in a text. By definition, the *entropy* of a source is the minimum average number of bits per source symbol required to describe the source symbols without loss.

Some examples illustrate that concept. First consider a sequence of fair coin flips. Because of their complete unpredictability, it is fairly intuitive that one needs 1 bit per symbol to describe the sequence.

Second, look at a sequence of 1 million 1s. This sequence can be described by the words "1 million 1s." These words require 12 characters, or 96 bits, to be encoded. That is, the sequence of 1 million 1s is encoded with 96 bits which amounts to about 0 bit per symbol.

Third, consider the sequence $\mathbf{Y} = \{X_1, X_1, X_2, X_2, X_3, X_3, \ldots\}$ of twice-repeated fair coin flips. Since it takes one bit per symbol $X_1, X_2, \ldots$, we see that we can describe the sequence $\mathbf{Y}$ with 0.5 bit per symbol.

More generally, we suspect that if there is some correlation between successive bits in a sequence, then we require fewer than 1 bit per symbol to describe the sequence.

Claude Shannon has shown that, under weak assumptions, one can define a quantity H, called the *entropy rate* of the source. This quantity is the minimum average number of bits per source symbol required in lossless compression. We explain the basic ideas of this theory in Section 8.10.

8.4.4 Finding the Minimum Number of Bits

How can one automatically use only the minimum average number of bits per symbol, or at least not much more that this minimum? Practical systems use a combination of clever tricks. Here is a partial list. We revisit these examples in subsequent sections.

- *Models* Fit model parameters and send the parameters. Examples: fractals, code excited linear predictors.
- *Prediction + error encoding* Predict next symbol, send error. Examples: video conference compression.
- *Differential encoding* Send difference with previous symbol. Examples: motion compensation, DPCM.
- *Run length encoding* Send length of run of repeated symbols. Examples: silence suppression, fax encoding.
- *Dictionary* Point to, do not repeat a previously seen string. Examples: Lempel-Ziv.
- *Short codes for likely symbols* Longer for rarer. Examples: Huffman (e.g., JPEG, MPEG).

We examine these basic building blocks of compression algorithms.

8.4.5 Huffman Encoding

The main idea of Huffman encoding is to use short codes for frequent symbols and longer ones for rare symbols. For example, consider a source that produces a string of symbols that take four possible values $\{A, B, C, D\}$ and that, in the long run, these symbols are produced in the following proportions: A: 45%, B: 45%, C: 5%, D: 5%. We are not making any independence assumption or using any model for the correlation of successive symbols.

Using the code

$$A\text{: } 00,\ B\text{: } 01,\ C\text{: } 10,\ D\text{: } 11,$$

one needs 2 bits per source symbol. However, using the code

$$A\text{: } 0,\ B\text{: } 10,\ C\text{:}110,\ D\text{: } 111, \tag{8.2}$$

we need $1(0.45) + 2(0.45) + 3(0.05) + 3(0.05) = 1.65$ bits per symbol. The code that we chose in (8.2) is prefix-free. This means that the first part of no codeword is another codeword. The code A: 0, B: 00, C: 1, D: 11 is not prefix-free because the first part 0 of the codeword 00 for B is another codeword (for A). A prefix-free code has the property that any concatenation of codewords can be decoded without ambiguity. Thus, for the code in (8.2), the string 1011001111001100 can only be decoded as $BCADBACA$. When using a code that is not prefix-free, one must insert special symbols to separate the source symbols, which increases the average number of bits per symbol and ends up transforming the code into a prefix-free code.

The code (8.2) is the prefix-free code that requires the smallest average number of bits per symbol for our example. This code is called the Huffman code. Before explaining the simple procedure that produces the Huffman code, we note that if the symbols are i.i.d. (independent and identically distributed), then we can compute the entropy of the source and we find that we need $H = -0.45 \log 0.45 - \cdots - 0.05 \log 0.05 = 1.47$ bits per symbol. Thus, the Huffman code does not achieve the Shannon bound. Of course, achieving or even approaching that bound is substantially more complicated than constructing the Huffman code. More about this when we study the Lempel-Ziv compression algorithm.

We now explain the general method for constructing the Huffman code. We are given K symbols $\{1, 2, \dots, K\}$ with the relative proportions in which they appear in a long string of symbols that the source produces: $\{p(1), p(2), \dots, p(K)\}$. Thus, $p(k)$ is the fraction of symbols that are equal to k. The numbers $p(k)$ are nonnegative and add up to 1.

We construct a tree recursively. At each step, we join the two nodes with the smallest values and we attach the sum of their values to their aggregate. Figure 8.5 shows this procedure for our previous example. We first join the symbols C and D because their values 0.05 and 0.05 are the two smallest. We join them by creating a subtree whose root is allocated the value sum 0.1 of the values of the two symbols. We then join this root and symbol B to create a new root with value 0.55. Finally, we join this new root and symbol A. The resulting tree defines a code as follows. The code that corresponds to a symbol is obtained by starting from the top root and traveling down toward the symbol. As we go down, each move to the left corresponds to a 0 and each move to the right corresponds to a 1. For instance, the codeword for symbol B is 10 because we first move right (and down), then we move left (and down) to reach B.

Figure 8.6 is another example. This algorithm produces a prefix-free code. Indeed, any path down the tree reaches a leaf only at the end of the path and not at any intermediate point along the path. It is not completely straightforward to show that this code has the minimum average length over all prefix-free codes.

FIGURE 8.5

Constructing a Huffman code.

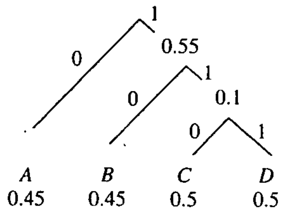
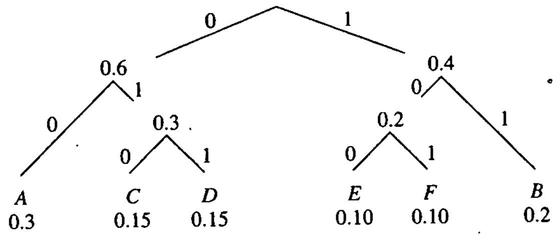


FIGURE 8.6

Another example of a Huffman code.



A: 00
B: 11
C: 010
D: 011
E: 100
F: 101

8.4.6 *Lempel-Ziv Compression*

This algorithm produces a code that achieves the Shannon limit. We should qualify this statement. Consider a source that produces random symbols $\{X_n, n \geq 1\}$. Under the ideal implementation, the average number of bits per symbol that the Lempel-Ziv algorithm requires to encode the first n symbols of the source approaches the entropy rate of the source.

What is remarkable about the Lempel-Ziv algorithm is that it does not require any prior information about the statistics of the source. Versions of this algorithm are implemented in most compression utilities.

Roughly speaking, this algorithm replaces a string by a pointer to the first occurrence of the string, if it has been seen before. For instance, the algorithm encodes the string

$$\text{a bit is a bit is a bit} \tag{8.3}$$

as

$$\text{a bit is } [1, 9][1, 5]. \tag{8.4}$$

The first part of the sentence "a bit is" does not contain pieces that appeared earlier in the sentence. Accordingly, this part must be transmitted by the code. (How else would the receiver know about it?) The next part "a bit is" has been seen previously. Instead of repeating this string, the algorithm replaces it by a pointer to the location of the start of that earlier instance of the string and by the length of that fragment. In our example, the earlier fragment starts in location 1 and has length 9 characters (including spaces). The last piece of the sentence, "a bit" has been seen earlier, starting at location 1 and with a length of 5 characters.

From the code (8.4), there is no ambiguity in reconstructing the sentence (8.3). Moreover, this reconstruction is straightforward. The encoding is more complex. It requires parsing the original sentence to locate fragments that have been seen previously. Encoding algorithms use a "dictionary." The source output is parsed and at each step, the shortest string that is not yet in the dictionary is added to it.

Figure 8.7 illustrates this parsing process. The parsing proceeds from left to right and starts with an empty dictionary. The first bit, 0, is a string that is not in the dictionary. We add that string and indicate that fact by inserting a comma after the first 0. The next shortest string that is not yet in the dictionary is 00. We add it to the list. The process

FIGURE 8.7

Parsing a bit string in the Lempel-Ziv algorithm.

Original string:

0 0 0 1 1 0 0 1 0 1 1 1 0 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 0 1

Parsing:

0, 0 0, 1, 1 0, 0 1, 0 1 1, 1 0 1, 0 0 1, 0 0 1 0, . . .

continues in this way. To compress the original string, we replace each substring produced by the parsing as follows. We strip the substring of its last bit. This shortened string must have appeared in the dictionary. We replace the substring by a pointer to the location of the shortened string in the dictionary together with the last bit of the substring. For instance, the string 101 is replaced by a pointer 4 to the shortened string 10 together with the last bit 1 of the string. As the process continues, the strings that the parsing produces get longer. A string of length n gets replaced by a pointer to the shortened string. This pointer requires a number of bits approximately equal to the logarithm in base 2 of the number of strings in the dictionary. Remarkably, the analysis shows that the process ends up using an average number of bits per symbol equal to the entropy rate of the source, at least if we have an infinite dictionary to run the algorithm.

8.5 Audio Compression

Audio compression schemes are used in applications that range from regular, cordless, and cellular telephone to audio for digital TV, CD-ROMs, Mini-Disc, Digital Compact Cassette, 3/2 Stereo, Internet Phone, RealAudio. Here is a partial list:

PCM	pulse code modulation
ADPCM	adaptive differential PCM
SBC	sub band coding
VSELP	vector sum excited linear prediction
CELP	code excited linear prediction
MPEG	Motion Picture Experts Group

Figure 8.8 shows the bit rates that these schemes produce. We explain some of these schemes below.

8.5.1 *Differential Pulse Code Modulation (DPCM)*

DPCM transmits differences $X_{n+1} - X_n$ between successive samples X_n instead of transmitting the sample values. Successive audio samples tend to be similar. Accordingly, their differences are small and can be quantized accurately with fewer bits than the samples.

Figure 8.9 illustrates DPCM. The figure shows a sequence of audio samples and their successive differences.

8.5.2 Adaptive DPCM (ADPCM)

Adaptive DPCM goes one step beyond DPCM. Instead of simply computing the difference $X_{n+1} - X_n$ between successive samples, ADPCM computes the difference $X_{n+1} - \hat{X}_{n+1}$ between the next sample X_{n+1} and the predicted value $\hat{X}_{n+1}$ of that sample based on samples seen so far. In practice, $\hat{X}_{n+1}$ may be computed from a linear combination of recent sample values. Intuitively, if recent samples are increasing, it may be reasonable to expect that the next one is yet a bit larger. Figure 8.10 shows an example of ADPCM.

8.5.3 Subband Coding ADPCM

The idea behind subband coding ADPCM is that the human ear has different sensitivities to different frequencies. It is then sensible to reproduce these different frequency bands

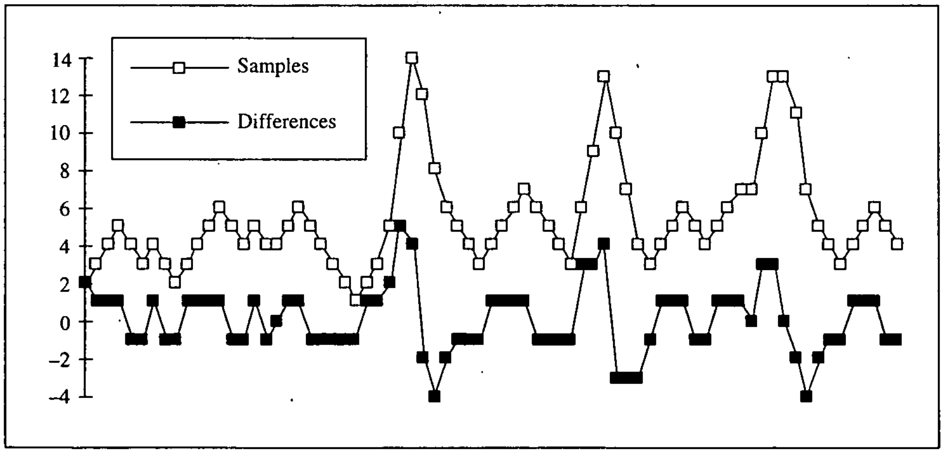
FIGURE 8.8

Bit rates produced by various audio compression algorithms.

Application	Schemes	Bit rates (kbps)
Speech	PCM	64
	ADPCM	40–64
	SBC	16–32
	VSELP-CELP	2.4–8
Audio	PCM	1400
	MPEG	128–384

FIGURE 8.9

Differential pulse code modulation.



with accuracies that correspond to these sensitivities. The basic block diagram of these algorithms is shown in Figure 8.11.

More complex algorithms combine masking with subband coding. That is, when the volume is much larger in one band than in the other, one may reduce the resolution of that weaker band or suppress it altogether. MPEG audio compression uses that mechanism.

8.5.4 *Code Excited Linear Prediction*

This method is designed to achieve very low bit rates, at the cost of a significant loss in quality, unfortunately. The block diagram of a CELP system is shown in Figure 8.12.

The speech production process is modeled as a filter that is excited by a collection of different codewords. For a given speech fragment, the algorithm selects the parameters of the filter and the codeword in the codebook that produce the best approximation of the fragment. The algorithm then transmits the filter parameter and the index of the codeword.

The decoder performs the reverse operations. It reproduces the speech fragment by exciting the filter with the codeword.

FIGURE 8.10

Adaptive differential pulse code modulation.

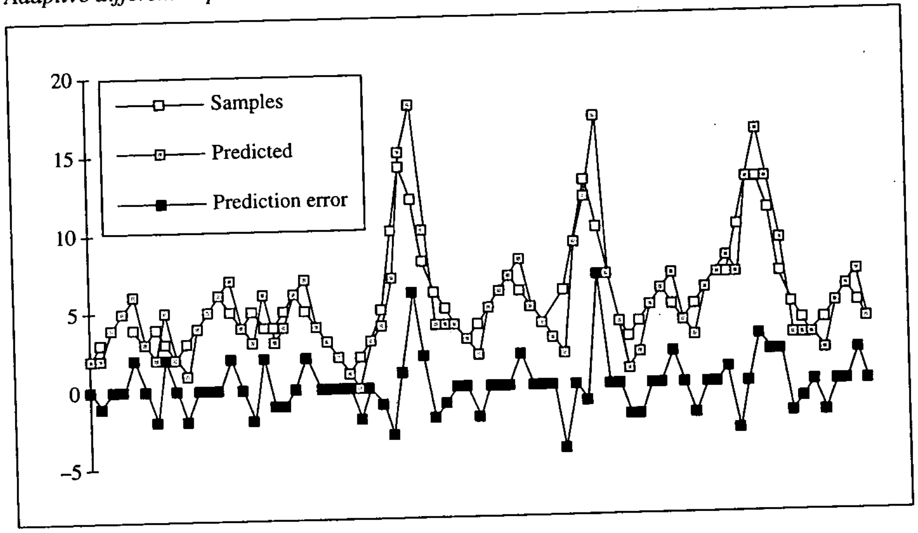


FIGURE 8.11

Subband coding ADPCM.

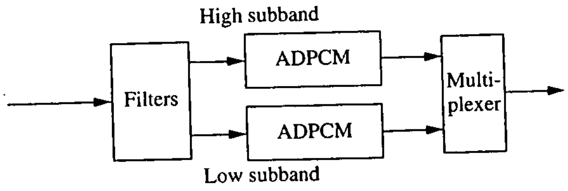
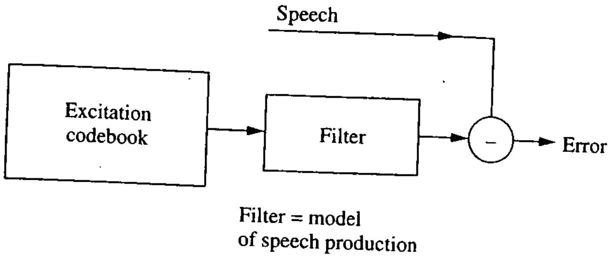


FIGURE 8.12

Block diagram of CELP system.



8.6 Video Compression

As with audio compression, we are only able to skim the surface of this subject. Video compression is necessary because video clips or signals require too many bits without compression. Fortunately, there is a lot of redundancy and of barely perceptible information in video, so that compression algorithms are very effective.

Without compression, we can estimate the bit rate needed to transmit a video signal as $360 \times 288 \times 8 \times 3 \times 24 = 60$ Mbps. The calculation is picture size times number of bits per color per pixel times number of colors times number of pictures per second. This is for a small image.

HDTV without compression requires about 8000 Mbps. With compression, we get excellent results with 20 Mbps.

8.6.1 Some Algorithms

Here are a few representative video compression algorithms (the numbers in parentheses indicate the applicable steps in the list below):[1]

- JPEG: pictures (1–3)
- H.261 ($p \times 64$): ISDN video conferences
- H.263 (≤ 64): video conferences (1–4)
- MPEG 1: 1.5 Mbps, DBS, CATV (1–4)
- MPEG 2: 10 Mbps and more, DBS, CATV, HDTV, DVC, DVD (1–4)
- MPEG 4: 64 kbps (not ready), model-based

These algorithms combine a number of clever ideas. The image is decomposed into blocks. Then the following steps are applied.

1. Transform image: DCT produces coefficients of the average value and of the higher-frequencies in the image

[1] JPEG = Joint Photographic Experts Group, ISDN = integrated services digital networks, DBS = direct broadcast satellites, CATV = community antenna TV, HDTV = high-definition TV, DVC = digital video cassette, DVD = digital video disc.

2. DCT coefficients are correlated. Accordingly, one calculates differences
3. Encode coefficients by Huffman encoding for each frame
4. Motion compensation between frames

We explain these basic ingredients below.

8.6.2 Discrete Cosine Transform

An image—more precisely, a block of pixels inside the image—is a matrix $[f(x, y)]$ of numbers that represent the level of light (luminance) of the pixels. (We think of a black and white picture, for simplicity.) The DCT of this block is the two-dimensional Fourier transform $[F(m, n)]$ of the matrix. One can appreciate the meaning of this transform if one knows that we can reconstruct the block from the transform by calculating

$$f(x, y) = \sum_{m,n} F(m, n) \cos(mx) \cos(ny). \tag{8.5}$$

That representation shows that the block is viewed as a superposition of product of sine wave. Figure 8.13 shows three images with their DCT. The left image has a constant luminance. Its DCT has only one nonzero coefficient $C = F(0, 0)$. Indeed, using (8.5) with $F(0, 0) = C$ and the other values of F equal to zero, we find $f(x, y) = C$, so that the luminance is constant across the picture.

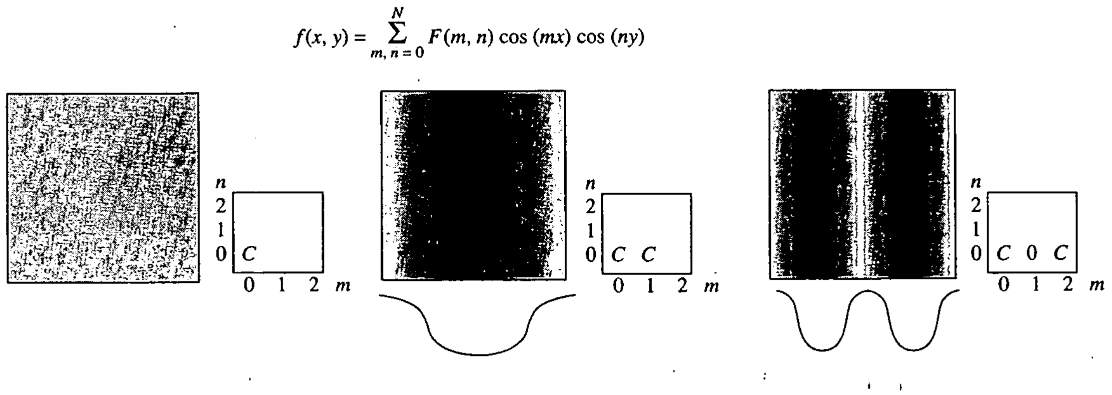
The center image has a DCT with two nonzero coefficients: $F(0, 0) = F(1, 0) = C$. Using (8.5) we find $f(x, y) = C + C\cos(x)$. The luminance of the image varies according to a sine wave along the horizontal axis of the image.

The image on the right can be understood similarly. By using more coefficients, one ends up decomposing the image into a sum of gridlike patterns. If the image luminance varies slowly, then only the low-order coefficient F (for small values of m and n) are nonzero.

One key idea is that the eye may not be very sensitive to very fine details, so that we may set to zero the coefficients $F(m, n)$ for large values of m and n that correspond to such fine details.

FIGURE 8.13

Three images and their DCT.



Another important observation is that video images—across a movie—tend to have similar characteristics when transformed by DCT. That is, the coefficients F tend to have comparable magnitudes across images. Consequently, Huffman encoding of these coefficients does not require learning the statistics for video applications. The situation is different for still images where learning the statistics (the fractions of coefficients of different blocks having specific values) improves the performance of the Huffman code considerably.

A third observation is that many DCT coefficients tend to be zero. A good encoding consists in counting the zeros instead of listing them. This method, called run-length encoding, is combined with Huffman encoding. Specifically, the Huffman encoding is applied to the length of zeros followed by the value of the first nonzero DCT coefficient.

8.6.3 Motion Compensation

Substantial gains are achieved in video compression by using the similarity of successive frames. A naive way of doing this would be to compute the difference between two successive frames.

Motion compensation is more subtle. It examines how each block in the image has moved from one frame to the next. Instead of transmitting the next block, one transmits the motion vector that best approximates the motion of the block plus the difference between the translated block and the actual block in the next frame. This idea is illustrated in Figure 8.14.

8.6.4 MPEG

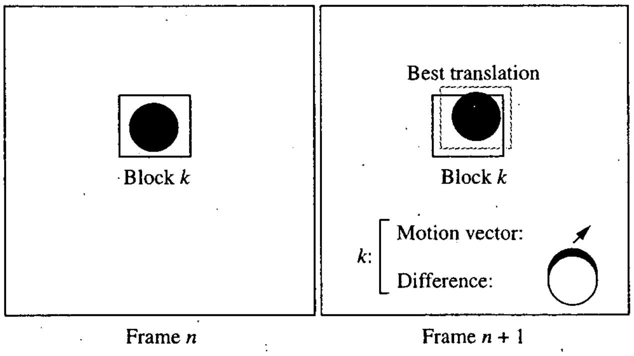
We conclude this section with a very brief look at MPEG, a family of video compression algorithms. MPEG produces a *group of pictures:*

$$I\ B\ B\ B\ P\ B\ B\ B\ P\ B\ B\ B\ P\ B\ B\ B\ P.$$

In this sequence, I designates an "intraframe" that has been compressed by itself, without any reference to other frames. The I frame is encoded using DCT with Huffman encoding

FIGURE 8.14

Motion compensation. A block is moved to find its best match in the next frame. The encoder sends the motion vector and the matching error.



of the run lengths of zeros followed by the value of the first nonzero DCT coefficients. These coefficients are listed in the zig-zag order $F(1, 0)$, $F(0, 1)$, $F(2, 0)$, $F(1, 1)$, $F(0, 2), \ldots$ to determine the run length of zeros and the first nonzero coefficient. Tables of these values and their Huffman code are standardized. For instance, $(0, 0, 0, 0, 1)$ is encoded as $(00110, 0)$ where the last zero indicates that the nonzero coefficient is positive. As other examples, $(0, 0, 0, 0, -1)$ is encoded as $(00110, 1)$ and $(0, 0, 0, 0, 2)$ as $(0000001111, 0)$.

The P designates a frame that has been predictive-coded, using motion compensation. The B designates a frame that has been bidirectionally predictive-coded. That is, the prediction is based on past and future I and P frames.

MPEG encodes the motion vectors differentially, because such vectors tend to be similar across a picture (think of a camera pan). Estimating the motion is the time-consuming part of the MPEG algorithm.

Going back to Figure 8.14, we see that estimating the motion consists in finding the translation of a given block in one frame that best approximates the corresponding block in the next frame. Ideally, one would like to explore all the possible translations left and right, up and down, of up to 6 picture elements. However, this amounts to calculating 169 times the difference, say sum of squares, of the blocks.

A number of clever approximate search algorithms have been developed. We give just one example to indicate what can be done instead of an exhaustive search. One algorithm first looks at the nine displacements $(-3, 0, +3) \times (-3, 0, +3)$. The block is then moved to the best of these nine displacements. In the second step, the algorithm explores the nine displacements $(-2, 0, +2) \times (-2, 0, +2)$ and moves to the best displacements. Finally, the algorithm explores the nine displacements $(-1, 0, +1) \times (-1, 0, +1)$. These three steps require calculating only 27 differences, a savings of a factor 6 for a comparable performance.

8.7 Complement 1: Secret Key Cryptography

In this section we discuss the most widely used secret cryptography codes. We then explain how Alice and Bob can agree on the secret functions E and D by communicating over a public channel.

8.7.1 *Secret Codes*

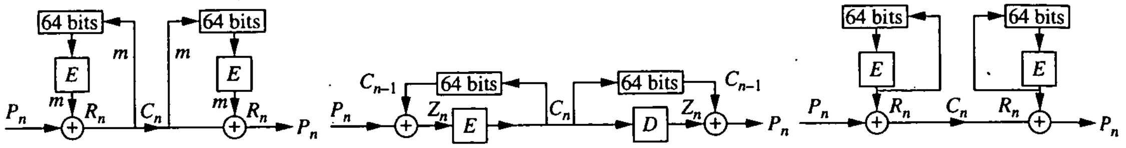
A few secret codes have withstood the attacks of many clever researchers and are believed to be good codes. We explain these codes. Specifically, we discuss DES and IDEA.

DES

One of the standards adopted for encryption uses a combination of substitution and transposition. In 1977, the National Bureau of Standards adopted the *Data Encryption Standard* (DES) as the official encryption standard for the unclassified information of the U.S. government. DES was developed by IBM in cooperation with the National Security Agency. There are inexpensive VLSI (very large scale integration) chips that perform the necessary encryption and decryption.

FIGURE 8.15

Encoding modes for large messages using DES or IDEA. From left to right: cipher feedback, cipher block chaining, and output feedback modes.



To perform the encryption of a text, DES first divides the text into blocks of 64 bits. Each block of 64 bits is then encoded separately into new blocks, also of 64 bits. Thus, DES is a mapping $C = E_K(P)$ where both C and P are words of 64 bits. The key K is a word of 56 bits.

The first step in the DES encoding of P is to calculate $P_1 = T(P)$, where T is a given bit permutation. The DES algorithm then calculates successively $P_{i+1} = F(P_i, K_i)$ for $i = 1, 2, \ldots, 16$, where the K_i are different subsets of 48 bits of K. The function F is also specified by the algorithm. Finally, the DES algorithm calculates $C = T^{-1}(P_{16})$ where T^{-1} is the inverse of the bit permutation T. The function D_K is calculated by the same algorithm where the order of the keys K_i is reversed.

The encryption E_K has the property that a modification of a single bit in P has about a 50 percent chance of modifying each bit in C and these bit modifications appear independent. It is thus very difficult to reconstruct P from C without knowing the key K.

A successor to DES is being developed and will probably be called AES, for Advanced Encryption Standard.

IDEA

In 1991, Xuejia Lai and James Massey proposed a new secret code called *International Data Encryption Algorithm*. IDEA is similar to DES but is based on a 128-bit key.

Large Messages

Three methods are used for encoding long messages with either DES or IDEA. The methods are shown in Figure 8.15.

The cipher feedback mode has the advantage of being self-synchronizing: no initial value has to be set in the registers. Moreover, an error in a block of m bits has limited error propagation. In most applications, $m = 1$, which avoids the problem of requiring correct alignment of the boundary of these blocks.

The output feedback mode can be thought of as a one-time pad where the pseudorandom sequence is generated by the output feedback.

8.8 Complement 2: Public Key Cryptography

We explain the RSA public key cryptography code.

8.8.1 RSA

The Rivest, Shamir, Adleman (RSA) public key system is based on the difficulty of finding the prime factors p and q of pq when they are large. The system also uses the following lemma:

Lemma
Let p and q be two prime numbers. Define $n = pq$, $z = (p-1)(q-1)$. Choose e such that d and z have no common divisor other than 1. Then choose d so that $ed = 1 \bmod z$ (i.e., ed is a multiple of z plus 1).

If $P \in \{0, 1, \dots, n-1\}$ and $C := P^e \bmod n$, then $P = C^d \bmod n$.

For a proof of this lemma, see Section 8.9.

For instance, let $p = 3, q = 11, n = 33, z = 20, d = 7$, and $e = 3$. The lemma implies that if $C = P^3 \bmod 33$, then $P = C^7 \bmod 33$ for all $P \in \{0, 1, \dots, 32\}$. As an example, if $P = 5$, then one verifies that $C = 26$ and that $26^7 \bmod 33 = 5$.

The lemma enables us to construct a public key system by making the pair of numbers $(e,\ n)$ public. To know the decoding function, an eavesdropper needs the pair $(d,\ n)$. However, finding d from $(e,\ n)$ is believed to be a hard problem.

8.9 Complement 3: Proof of RSA Lemma

Here is a proof of the lemma.

Proof of Lemma
Since $C := P^e \bmod n$, as we showed in the verification of (8.1), we know that

$$C^d = P^{ed} \bmod n. \tag{8.6}$$

We show below that

$$P^z = 1 \bmod n. \tag{8.7}$$

This equality implies that (modulo n), $P^{ed} = P^{kz+1} = P$.

It remains to show (8.7). We prove it assuming that P and n have no common factor other than 1. In other words, we assume that P is not a multiple of p nor of q. (The result holds without that assumption.)

To do this, define for $m \geq 1$ the set $F(m)$ of all the integers in $\{1, 2, 3, \dots, m-1\}$ that are coprime with m. Two numbers a and b are coprime (or relatively prime) if their greatest common divisor gcd$(a,\ b)$ is equal to 1.

Using Euclid's algorithm one can check that a and b are coprime if and only if $ax + by = 1$ for some integers x and y. To see this, assume $a < b$ and set $x(1) = 1, y(1) = 0, x(0) = 0, y(0) = 1$ so that $a = q(1)$ and $b = q(0)$ with

$$q(n) = ax(n) + by(n).$$

Next, note that if

$$q(n) = ax(n) + by(n) < q(n-1) := ax(n-1) + by(n-1) = kq(n) + r(n),$$

where k is the quotient of $q(n-1)$ by $q(n)$ and $r(n)$ is the remainder, then $\gcd(q(n-1), q(n)) = \gcd(q(n), r(n))$. Moreover,

$$r(n) = ax(n+1) + by(n+1)$$

with

$$x(n+1) = x(n-1) - kx(n), y(n+1) = y(n-1) - ky(n).$$

Continuing in this way, we reach some iteration n with $r(n) = 0$ and it must then be that $q(n) = 1$. Indeed, otherwise $q(n)$ would be the gcd of $q(n-1)$ and $q(n)$ and therefore of $b = q(0)$ and $a = q(1)$. But $q(n) = 1$ shows that $ax + by = 1$ with $x = x(n)$ and $y = y(n)$.

We use that characterization of coprime numbers to show that $F(n)$ is closed under multiplication modulo n. That is, if $a, b \in F(n)$, then $a.b \bmod n \in F(n)$. Indeed $ax + ny = 1 = bx' + ny'$, so that

$$1 = (ax+ny)(bx'+ny') = ab(xx') + n(ybx' + axy' + nyy'),$$

which shows that ab and n are coprime.

Next we show that if $a \in F(n)$, then there is some $b \in F(n)$ so that $ab = 1 \bmod n$. Indeed, a and n are coprime so that $ax + ny = 1$ for some (x, y) so that we can choose $b = x$.

Now, assume that $a, b, c \in F(n)$ with $a \neq b$. We claim that $ac \neq bc$ mod n. To see this, let's multiply both sides of the equality by h such that $hc = 1 \bmod n$. We find $ach = bch \bmod n$, so that $a = b \bmod n$.

From the above, it follows that $\{ab, b \in F(n)\} = F(n)$.

Let us denote by ρ the product of all the elements of $F(n)$ and by σ the number of those elements. Then, multiplying all the elements of $\{ab, b \in F(n)\}$ together we should get $a^{\sigma}\rho \bmod n$, but we should also get ρ since that set is $F(n)$. Hence,

$$a^{\sigma}\rho = \rho \bmod n.$$

Let δ be such that $\delta\rho = 1 \bmod n$. Multiplying both sides of the above equality by δ we find that

$$a^{\sigma} = 1.$$

It remains to show that $\sigma = z$ to demonstrate that $a^z = 1$ for any element a of $F(n)$. Remember that $z = (p-1)(q-1)$. It is immediate that $F(p) = \{1, 2, \ldots, p-1\}$ so that the cardinality (number of elements) $|F(p)|$ of $F(p)$ is $p-1$. Similarly, $|F(q)| = q-1$. Recall that $n = pq$. To show $|F(n)| = (p-1)(q-1)$, it suffices to show that each $a \in F(n)$ corresponds to a unique pair (x, y) with $x \in F(p)$ and $y \in F(q)$ such that

$$a = x \bmod p = y \bmod q. \tag{8.8}$$

Assume that (8.8) holds with $x \in F(p)$ and $y \in F(q)$. Then

$$xu + pv = 1, a = x + kp \rightarrow (a - kp)u + pv = 1 \rightarrow (a, p) \text{ coprime.}$$

Similarly, (a, q) coprime. Hence, (a, pq) coprime so that $a \in F(n)$.

Conversely, assume $a \in F(n)$ and that (8.8) holds. Then $au + pqv = 1$, so that by (8.8),

$$a = kp + x \rightarrow (kp + x)u + pqv = 1$$

which implies that (x, p) are coprime. Similarly, (y, q) are coprime. This concludes our proof.

8.10 Complement 4: Source Coding Theory

In this section we outline the source coding theory of Claude Shannon. Shannon showed that it is possible to quantify information. His starting point is that information describes an unpredictable outcome. For instance, the outcome of a fair coin has two equally likely values "head" and "tail." By convention, we agree that it takes one bit of information to describe this outcome.

If the outcome of a random experiment is equally likely to take any value in a set with 2^n elements, then we can identify this outcome with a string of n fair coin flips and we find that this outcome requires n bits to be described.

More generally, if a string of n symbols is equally likely to take any value in a set with 2^{nH} elements, it takes nH bits to describe the n symbols, or H bits per symbol. For example, consider the string (X, Y, Z) produced by a source and assume that the string is equally likely to be any one of the following four strings: 000, 100, 010, and 001. In this case, we find that it takes 2 bits to describe the three symbols, or $\frac{2}{3}$ bit per symbol.

The key observation of Shannon is that for most sources, the sequence of symbols it produces must be "typical" and there are few typical sequences!

We first explain this observation in the case of i.i.d. (independent and identically distributed) symbols. Let then $\{X_n, n \geq 1\}$ be i.i.d. with

$$P(X_n = k) = p(k), k = 1, 2, \ldots, K$$

where the numbers $p(k)$ are positive and add up to one. Fix $n \gg 1$ and consider $\mathbf{X} = (X_1, X_2, \ldots, X_n)$. Define

$$g(\mathbf{x}) := g(x_1, \ldots, x_n) = \log p(x_1) + \cdots + \log p(x_n), \text{ for } x_i \in \{1, \ldots, K\}, 1 \leq i \leq n.$$

In the above definition, log designates the logarithm in base 2.

Note that, by the law of large numbers, $g(\mathbf{X})/n = [\log p(X_1) + \cdots + \log p(X_n)]/n \approx -H$ where

$$H := -E \log p(\mathbf{X}) = -p(1) \log p(1) - \cdots - p(K) \log P(K). \qquad (8.9)$$

Consequently, the random sequence $\mathbf{X}$ is in a set $\mathbf{S}$ of sequences defined as follows:

$$\mathbf{S} := \{\mathbf{x} = (x_1, \ldots, x_n) \text{ such that } g(\mathbf{x}) = -nH\}.$$

But, for any $\mathbf{x}$ in $\mathbf{S}$,

$$P(\mathbf{X} = \mathbf{x}) = p(x_1) \cdots p(x_n) = 2^{g(\mathbf{x})} \approx 2^{-nH}.$$

Consequently, the set **S** has $|\mathbf{S}|$ elements where

$$|\mathbf{S}| = 2^{nH}.$$

Moreover, **X** is equally likely to take any value in **S**. Accordingly, we conclude that it takes nH bits to describe this random sequence. That is, it takes H bits per symbol to describe the sequence, as we wanted to show.

As a simple example, consider a sequence of flips of an unfair coin such that $P(\text{head}) = 1 - P(\text{tail}) = p$. Using (8.9) we find that for this sequence,

$$H = H(p) := -p \log p - (1-p) \log (1-p).$$

This entropy, called the *binary entropy*, is shown in Figure 8.16. As the figure shows, when $p = 0.5$, it takes 1 bit per symbol to describe the outcome of a sequence of coin flips. However, if $p \ll 1$, then it takes much fewer bits to describe the sequence of coin flips because this sequence typically contains very few 1s and there are few such sequences.

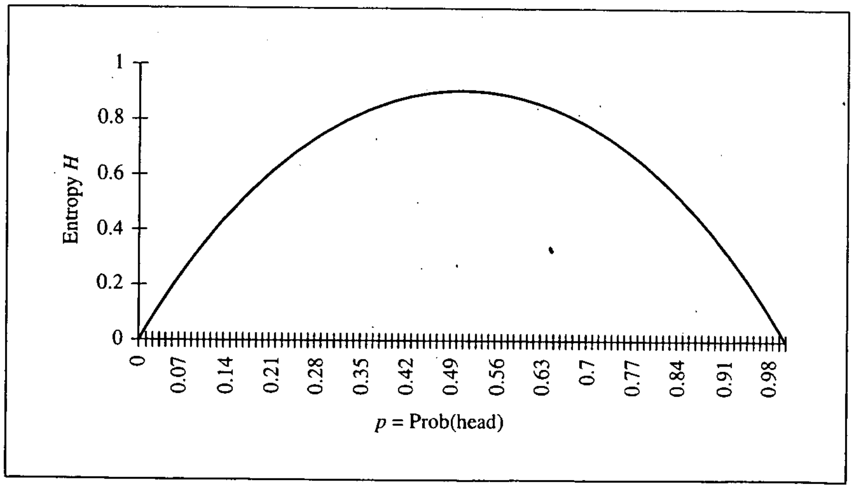
This theory extends to non-i.i.d. symbols. In that case, the entropy of the source is H bits per symbol where H is the average entropy of X_{n+1} given the past symbols. For instance, if $\{X_n, n \geq 1\}$ is an irreducible Markov chain on $\{1, 2, \ldots, K\}$ with transition probabilities $P(i, j) = P[X_{n+1} = j | X_n = i]$ and invariant distribution $\pi = \{\pi(i), i = 1, 2, \ldots, K\}$, then

$$H = -\sum_i \pi(i) \sum_j P(i, j) \log P(i, j). \tag{8.10}$$

As an illustration of this formula, consider the Markov chain with the transition diagram shown in Figure 8.17. This source produces random strings with symbols in $\{1, 2, 3\}$. After it outputs the symbol 1, the source outputs the symbol 2 with probability $P(1, 2) = 0.4$ and the symbol 3 with probability $P(1, 3) = 0.6$. Similarly, after it outputs the symbol 2, the source outputs the symbols 1 with probability $P(2, 1) = 0.5$ and the symbol 3 with probability $P(2, 3) = 0.5$. After symbol 3, the source produces the symbol 1 with

FIGURE 8.16

Binary entropy.



that it takes
. to describe

: $P(\text{head}) =$

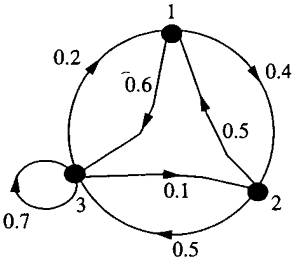
hows, when
f coin flips.
of coin flips
quences.
source is H
For instance,
probabilities
$, 2, \ldots, K\},$

(8.10)

ion diagram
$2, 3\}$. After
$1, 2) = 0.4$
the symbol
e symbol 3
mbol 1 with

FIGURE 8.17

A Markov source..



probability $P(3, 1) = 0.2$, the symbol 2 with probability $P(3, 2) = 0.1$, and the symbol $P(3, 3) = 3$ with probability 0.7. The transition probabilities $P(i, j)$ define the matrix $\mathbf{P}$ given below:

$$\mathbf{P} = \begin{bmatrix} 0 & 0.4 & 0.6 \\ 0.5 & 0 & 0.5 \\ 0.2 & 0.1 & 0.7 \end{bmatrix}.$$

The invariant distribution π is the vector $(\pi(1), \pi(2), \pi(3))$ of nonnegative numbers that are such that

$$\pi\mathbf{P} = \pi \text{ and } \pi(1) + \pi(2) + \pi(3) = 1.$$

Solving these equations one finds that $\pi \approx (0.204, 0.146, 0.650)$. Using formula (8.10) we then compute the value of H and we find that

$$\begin{aligned} H = & \; 0.204(0.4 \log 0.4 + 0.6 \log 0.6) - 0.146(0.5 \log 0.5 + 0.5 \log 0.5) \\ & -0.650(0.2 \log 0.2 + 0.1 \log 0.1 + 0.7 \log 0.7) \\ \approx & \; 1.1. \end{aligned}$$

That is, it takes approximately 1.1 bit per symbol to describe the output of this random source. For example, we would need about 1100 bits to describe the first 1000 symbols produced by the source.

Problems

Problems with a * are somewhat more challenging. Those with a ^{c} are based on material in a complement. Problems with a ^{1} are borrowed from the first edition.

^{1}1. Consider the data encryption scheme. What property should the function F possess in order for the scheme to be decipherable by the receiver? Formulate the decoding algorithm that the receiver should use, assuming that the receiver knows the key K.

^{1}2. Assume that if the key K is known, then it takes 100 ms to decipher a message that has been coded using DES. Suppose that an eavesdropper chooses keys at random and tries to decipher the message. What is the average time required for the correct deciphering? (There are 2^{56} possible keys.)

^{1}3. The public key cryptography system is based on the idea that large prime numbers p and q (typically over 10^{100}) will be used. (This is why it is important to know many large prime numbers—it is here that number theorists are needed!) If small prime numbers are used, then deciphering is relatively easy. Here is an example: Suppose a user transmits a message which is an integer between 1 and 10, using $n = 15$ and $e = 3$. The coded message yields the integer 8. What number was transmitted?

^{1}4. Show that the public key cryptography system can also be used as an electronic signature. (*Hint:* Check that the encoding function is invertible.)

^{1}5. Assume that, in a text, the number of consecutive 0s, X, is equally likely to be any integer between 1 and n. Derive an expression for the compression factor r_n achieved by run length encoding. Compute r_n for $n = 5$ and $n = 10$. Using Stirling's approximation for $n!$ ($n! = \sqrt{2\pi n}e^{-n}n^n$, for large values of n), find an asymptotic expression for r_n. The assumption that X is uniformly distributed is not very reasonable; for instance, in a text one is very likely to find large sequences of 0s due to margins, empty lines, etc. Will that unequal distribution improve r_n?

^{1}6. Is it better to use differential interline encoding to transmit a file containing random numbers representing the outcomes of a coin-tossing computer-generated experiment or a file containing the binary encoding of a picture of the sunset over Oahu? Why?

7. Construct the Huffman code for transmitting messages constructed by using the symbols A, B, C, D, and E occurring with frequencies 0.60, 0.30, 0.05, 0.03, and 0.02, respectively. What is the compression factor, defined as the number of bits per symbol in a straightforward binary encoding divided by the average number of bits per symbol of the Huffman encoding?

^{c}8. Assume that the entropy rate of the English language is about 0.1 bits per symbol. (This value is hypothetical.) How many bits do we need for a message digest $H(P)$ to be effective for plaintexts of up to 2000 characters?

9. Consider a prefix-free encoding used to encode a string of symbols. Explain why a single bit error in the encoded string can corrupt the decoding of the subsequent symbols.

^{c}10. Calculate the entropy rate of the Markov source which produces the symbols 0 and 1 according to the transition probabilities $P(0, 0) = 0.7 = 1 - P(0, 1)$ and $P(1, 0) = 0.4 = 1 - P(1, 1)$.

^{c}11. Consider the following simplified model of a digitized audio signal. The successive samples X_n take values in $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. If $X_n > 0$, then $X_{n+1} = X_n + 1$ with probability 0.4 and $X_{n+1} = X_n - 1$ with probability 0.6. Symmetrically, if $X_n < 0$, then $X_{n+1} = X_n - 1$ with probability 0.4 and $X_{n+1} = X_n + 1$ with probability 0.6. Finally, if $X_n = 0$, then $X_{n+1} = 1$ with probability 0.5 and $X_{n+1} = -1$ with probability 0.5. Estimate the entropy rate of this signal.

References

The material on security was developed in collaboration with Dr. B. Preneel.

An excellent reference on cryptography is Menezes (1997). Kaufman (1995) provides a good introduction to security issues and solutions. The book also explains the cryptography

ıe numbers p
know many
nall prime
le: Suppose a
= 15 and
:mitted?
ectronic

y to be any
or r_n achieved
.ng's
asymptotic
very
ces of 0s due

ıing random
:d experiment
ahu? Why?
.sing the
0.03, and
er of bits per
nber of bits

er symbol.
ligest $H(P)$

plain why a
osequent

nbols 0 and 1
ıd

e successive
$_{n+1} = X_n + 1$
ically, if
with
ınd
l.

95) provides a
cryptography

algorithms. In Ahuja (1996) you will find an accessible description of security systems. A discussion of security problems raised by Java can be found in McGraw (1997). For a discussion of message authentication codes and their properties, see Preneel (1995).

Two particularly enjoyable presentations of source coding, as well as other topics, can be found in Feynman (1996) and Rényi (1984). Once again, Shannon (1998) is *the* reference on information theory.

Vetterli (1995) is an excellent presentation of source coding, including multiscale representations using wavelet transforms.

The Lempel-Ziv algorithm is explained in Ziv (1997). The MPEG compression standard is described in LeGall (1991).

Bibliography

Ahuja, V., *Network and Internet Security,* Academic Press, 1996.

Feynman, R. P., *Lectures on Computation,* edited by J. G. Hey and R. W. Allen, Addison Wesley, 1996.

Kaufman, C., Perlman, R., and Speciner, M., *Network Security—Private Communication in a Public World,* Prentice-Hall, 1995.

LeGall, D., "MPEG: A video compression standard for multimedia applications," *Communications of the ACM,* 34, no. 4, pp. 46–58, April 1991.

McGraw G., and Felten, E. W., *Java Security,* John Wiley & Sons, 1997.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S., editors, *Handbook of Applied Cryptography,* CRC Press, 1997.

Preneel, B., and van Oorschot, P. C., "MDx-MAC and building fast MACs from hash functions," *Proc. Crypto '95, LNCS* 963, D. Coppersmith, ed., Springer-Verlag, 1–14, 1995.

Rényi, A., *A Diary on Information Theory,* John Wiley & Sons, 1984.

Shannon, C., "A mathematical theory of communication," *Bell System Technical Journal,* 27, pp. 379–423 and 623–656, 1948.

Vetterli, M., and Kovačević, J., *Wavelets and Subband Coding,* Prentice-Hall, 1995.

Ziv, J., and Lempel, A., "A universal algorithm for sequential data compression," *IEEE Trans. Information Theory,* 23, pp. 337–343, 1977.